Ontology for Ontology Evaluation

@Participants:

- Today (March 31, 2013) - fleshing out the diagrams and formalizing the ontology.
- Audio will be available on the “joinconference” line, as per usual ontolog virtual meets
- If you’re looking to see if material you would like to include has already been added / is being edited, I suggest using search (Ctrl+F) to search for the title of the talk / paper / author and jump to the relevant section below.

Technical requirements

for use in Purple Semantic Media Wiki (PSMW), to support organization, presentation, and reuse of OntologySummit2013 materials and content (i.e., for use by HC-04 team).

Coverage requirements

from various materials presented during the summit, especially those that identify:

- evaluable characteristics of ontologies,
- metrics and methods for such evaluation,
- lifecycle stages
- maturity models of ontologies and/or evaluation
- useful faceting characteristics of ontology characteristics
- characterizing ontology internally,
- characterizing ontology externally,
- characterizing relation of ontology specifics to particular use or application context
- context dependence of relevance of characteristic to quality/suitability
- context independence of relevance of characteristic to quality/suitability
- automatability ...
- ...

Initial High Level Conceptual Model

An **Ontology** is an entity that is generally realized as a (possibly singular) set of **artifacts** within an **organizational context**.
An Ontology should have a defined **scope**,
An Ontology should have a **intended use**
An Ontology should have **stakeholders**, ...
An Ontology can be **evaluated** according to a set of **criteria** (defined by **specifications** and **requirements**?), which can be divided along a **continuum** with two poles: **Intrinsic** and **Extrinsic**.
An ontology evaluation criteria is defined by specifications and requirements.
An ontology evaluation criteria may be intrinsic, extrinsic, or something between.
Each evaluation event/activity has an accompanying **method**.
There is a broader **context** which **motivates** the **creation** of the ontology
There is a broader **context** which **motivates** the **team** which creates an ontology.

An ontology also has a particular **lifecycle**, with a set of **phases** or **stages**, each of which may have particular evaluation criteria and characteristics which may **apply**.

Additionally, the **purpose** and **intended use** of an ontology may imply that it has a set of characteristics, may **select a subset of evaluation criteria**.
the **purpose** and **intended use** of an ontology may **select a subset of evaluation criteria**

Moreover, there exist a number of **metrics** which can be deployed, which provide **visibility** into ontology characteristics.

Characteristics, specification and requirements may also be **hierarchically** connected within themselves.

Should xref with: http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2011_ApplicationFramework_Synthesis
http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2011_ValueMetrics_Synthesis

Compendium of Various Evaluation Criteria

===

Denny Vrandeˇci´c, “Ontology Evaluation” in Handbook of Ontologies (2009)

The below is an excerpt from pages 295-296

- **Accuracy** [38]:
    - Do the axioms comply to the expertise of one or more users [17]?
    - Does the ontology capture and represent correctly aspects of the real world [38]?
- **Adaptability** [38]:
    - Does the ontology anticipate its uses?
    - Does it offer a conceptual foundation for a range of anticipated tasks?
    - Can the ontology be extended and specialized monotonically, i.e. without the need to remove axioms?
    - How does the ontology react to small changes in the axioms [19]?
    - Does the ontology comply to procedures for
        - extension, [17]? (named as extendibility by [22])
        - integration [17]?
        - adaptation [17]?
        - (also named expandability [19]?
        - also named sensitiveness by [19]?,
        - also named flexibility by [17])
- **Clarity** [22]:
    - Does the ontology communicate effectively the intended meaning of the defined terms?
    - Are the definitions objective and independent of context?
    - Does the ontology use definitions or partial descriptions?
    - Are the definitions documented?
    - Is the ontology understandable?
    - (also named cognitive ergonomics, [17]
    - (also named transparency [17])
    - (also named intelligibility [38])
- **Completeness** [19]/competency [23]:
    - Is the domain of interest appropriately covered?
    - Are competency questions defined?
    - Can the ontology answer the competency questions?
    - Does the ontology include all relevant concepts and their lexical representations?
    - (also called richness [38])
    - (also called granularity [38])

- **Computational efficiency** [17, 38]:
    - How easiy can reasoners process the ontology?
    - How successfully can reasoners process the ontology?
    - How fast can the usual reasoning services (satisfiability, instance classification, querying, etc.) be applied to the ontology?
- **Conciseness** [19]:
    - Does the ontology include irrelevant axioms with regards to the domain to be covered (i.e. a book ontology including axioms about African lions)?
    - Does the ontology include redundant axioms?
    - Does it impose a minimal ontological commitment [22], i.e. does it specifying the weakest theory possible and define only essential terms?
    - How weak are the assumptions regarding the ontology's underlying philosophical theory about reality [38]?
- **Consistency** [19]/coherence [22]:
    - Do the axioms lead to contradictions (logical consistency)?
    - Are the formal and informal descriptions of the ontology consistent, i.e. does the documentation match the specification?
    - Does the translation from the knowledge level to the encoding level show a minimal encoding bias?
    - Are any representation choices made purely for the convenience of notation [22]?
    - Are any representation choices made purely for the convenience of implementation [22]?
    - (covers also meta-level integrity, i.e. following ordering principles [17] like OntoClean (see chapter "An Overview of OntoClean")
- **Organizational fitness** [17]/commercial accessibility:
    - Is the ontology easily deployed within the organization?
    - Do ontology-based tools within the organization put constraints upon the ontology?
    - Was the proper process for creating the ontology used?
    - Was it certified, if required?
    - Does it meet legal requirements?
    - Is it easy to access?
    - Does it align to other ontologies already in use?
    - Is it well shared among potential stakeholders?

===

Obrst, Track A Intrinsic Synthesis (email by Leo Obrst)

- Is the ontology free of obvious inconsistencies and errors in modeling?
- Is the ontology structurally sound?
- How do we gauge that the ontology is structurally sound?
- Is the ontology appropriately modular?

- Is the ontology designed according to sound principles of logical analysis?
- Is the ontology implemented according to sound principles of logical analysis?
- Is the ontology designed according to sound principles of semantic analysis?
- Is the ontology implemented according to sound principles of semantic analysis?
- Is the ontology designed according to sound principles of ontological analysis?
- Is the ontology implemented according to sound principles of ontological analysis?
- Which intrinsic aspects of ontology evaluation are of greater value to downstream extrinsic ontology evaluation?

Structural Intrinsic Evaluation

Ontology evaluation that does not depend at all on knowledge of the domain being modeled. In general, structural intrinsic criteria are focused only on domain-independent notions, mostly structural, and those based on the knowledge representation language.

Draws upon mathematical and logical properties such as

- graph-theoretic connectivity,
- logical consistency,
- model-theoretic interpretation issues,
- inter-modularity mappings
- inter-modularity preservations

Structural properties such as

- branching factor of ontology constructs, are intrinsic structural properties
- density of ontology constructs, are intrinsic structural properties
- counts of ontology constructs, are intrinsic structural properties
- averages of ontology constructs,are intrinsic structural properties
- (meta-properties) transitivity of ontology constructs
- (meta-properties) symmetry of ontology constructs,
- (meta-properties) reflexivity of ontology constructs
- (meta-properties) equivalence of ontology constructs

Some examples of tools and methodologies that address **intrinsic ontology evaluation**:

- **Oops! Evaluation** web site at http://oeg-lia3.dia.fi.upm.es/oops/index-content.jsp and described by MariaPovedaVillalon
- **OntoQA** to develop metrics for any ontology based on structural properties and instance populations, described by SamirTartir
- **PatrickLambrix's** debugging of Isa-a taxonomic structures, especially with mappings between ontologies

Some examples of tools and methodologies that address **domain intrinsic ontology evaluation:**

- **OQuaRE** framework described by AstridDuqueRamos

- **OntoClean** (Guarino and Welty)
- **MariaCopeland**: Ontology Evolution and Regression Testing
- **MelissaHaendel**: Ontology Utility from a biological viewpoint
- **EdBarkmeyer**: Issues with mapping vocabularies (especially code-lists) to ontologies.

Domain Intrinsic Evaluation

Evaluation where some understanding of the domain is needed. Domain knowledge and better ways to represent that knowledge do come into play here, though divorced as much as possible from application-specific domain requirements that come more explicitly from extrinsic evaluation issues.

Domain Intrinsic Evaluation to determine that

- particular modeling constructs are in alignment with the reality it is supposed to model.
- (meta-properties)rigidity of modeling constructs,
- (meta-properties)identity of modeling constructs,
- (meta-properties)unity of modeling constructs
- Sources of suggestions to gauge the quality of the axioms of the ontology,:
    - metaphysics,
    - philosophical ontology,
    - semantics,
    - philosophy of language
- including e.g., the subclass/isa taxonomic backbone of the ontology and other structural aspects of the ontology.

Most of the aspects of Domain Intrinsic Evaluation focus on

- ontological content methods
    - better ontological analysis
    - better semantic analysis,
    - better meta-property analysis (such as provided by methodologies like OntoClean, etc.)

At the extrinsic edge of domain intrinsic evaluation, the context-independent measures from Structural Intrinsic evaluation begin to blend into the very context-dependent, application issues of Extrinsic evaluation.

Some examples of tools and methodologies that address domain intrinsic ontology evaluation:

- **OQuaRE** framework described by AstridDuqueRamos
- **OntoClean** (Guarino and Welty)
- **MariaCopeland**: Ontology Evolution and Regression Testing
- **MelissaHaendel**: Ontology Utility from a biological viewpoint
- **EdBarkmeyer**: Issues with mapping vocabularies (especially code-lists) to ontologies.

Extrinsic Evaluation Aspects:

Ontology evaluation where the structure and design of the ontology is opaque to the tester, and the evaluation is determined by the correctness of answers to various interrogations of the model. In general, application requirements and domain requirements that are specifically needed by particular applications are the focus of extrinsic evaluation.

Evaluation Across the Ontology Lifecycle

=====

West - Track C Synthesis (email)

We focus in the communique on the evaluation of ontologies under the following aspects

- Is the domain represented appropriately (given the requirements of the IT system)?
- Is the ontology human-intelligible?
- Is the ontology maintainable?
- Does the query/reasoning capability and performance meet the requirements of the IT system?

For each of the main kinds of ontology evaluation, it should highlight

- desirable characteristics of ontologies (e.g., reusability)
- measurable metrics (e.g., natural language definitions of classes and relations) linked to the desirable characteristics of the ontology.

This communique should not strive for an exhaustive list, but should focus on the most important characteristics. [track A, track B]

- A better understanding of the relationships between requirements at different levels
- A better understanding of how low level requirements support higher level requirements.
- Ontology development methodologies that align with and recognize similar stages to information systems development
- Ontology development methodologies with distinct
    - conceptual stages,
    - logical stages
    - physical stages,
- Ontology development does not start at the physical level with the choice of an implementation language.
- A clearer understanding of the architecture of ontology development
- A clearer understanding of the different aspects of architecture that are relevant
    - ontological commitments
    - language choices.

The development process for an ontology needs to have a number of stages, just like the data model in a traditional information systems development process.

Similarly requirements need to be identified in levels

- the capabilities of the overall system that the ontology is a component of,
- to capabilities of the ontology itself in that overall system,
- to high level requirements of the ontology like consistency
- to detailed requirements, like conforming to naming standards.

The ontology development needs to go through stages to match the stages in data model development in information systems.

- conceptual data model development in information systems
- logical data model development in information systems
- physical data model development in information systems.

There are architectural decisions to be made in terms of the choices of ontological commitments the ontology needs to make and does make. There are choices of ontology language and implementation environment. There is little evidence of this in current practice, where ontology development seems to start with someone writing some OWL or CL.

===

Track D Synthesis (in email)

Track D, as “Software Environments for Evaluating Ontologies”, falls within the current Communique outline in:
C. The State of the Art of Ontology Evaluation
(4) What tool-support is currently available to support the evaluation of the characteristics identified in C-2 and the best practices identified in C-3?

Within this vein, some preliminary Track D concepts that may be developed for inclusion in the Summit communique are, in no special order:

1. The notion of tool support of quality is broader than the track's title and should include "guidance" as well as "evaluation" of those ontology characteristics determining an ontology's quality and fitness. Ontology tools and software environments may intentionally constrain or recommend to the user proper ontology structure and content.
2. Tools may contribute this "evaluation" or "guidance" function at different points along the ontology life cycle, and for a given characteristic, some tools may perform better in one life cycle phase than in another phase where a different tool is better suited. Generally, appreciation of the full cycle of life of an ontology is not well established within the ontology community.
3. There are central aspects of ontology that may not be amenable to software control or assessment. For example, the need for clear, complete, and consistent lexical definitions of ontology terms is not presently subject to software consideration beyond identifying where lexical definitions may be missing entirely. Another area of quality difficult for software determination is the semantic fitness of an ontology to its world domain (reality) or to its application domain. Software guidance may be available for the fitness of candidate ontologies for import and reuse, but not so
4. The design, implementation, and use requirements of an ontology may affect how quality and fitness on a particular ontology characteristic are determined, as well as interpreted and valued. Perhaps all quality and fitness assessments by software should be traceable to stated ontology requirements.
5. Significant new ontology evaluation tools are currently becoming available to users. Carving a link between such tools and existing IT architecture and design tools (e.g., EA and SA) remains a future possibility in order to integrate ontology into mainstream application software development within enterprise or more focused IT environments. This capability could offer a definitive means of connecting ontology quality/fitness characteristics and measures to use case and application software requirements.
6. Approximate lexical and structural matching of a new ontology or ontology component to the content of a repository of known ontologies may offer an effective means of identifying comparable ontology content for: 1) demonstrable coding patterns; 2) confirmation of authoring approach; and 3) identification of reuse candidates.
7. Given sufficient results from the Ontology Quality Software Survey, the degree to which current tool capabilities align with ontology quality priorities expressed by Tracks A-C.
8. Discoveries about the state of ontology evaluation stemming from the Hackathon and Clinic experiences.

======

LifeCycle Summary by FabianNeuhaus (in email response)

1. Rationale: Why do you need an ontology?
2. Analysis 1 (Competency Questions)

    a. Bottom-Up: What are semantics of current data sources?
    b. Top-Down: What would you like to ask?
3. Analysis 2
    a. What are the referents, concepts: entities, relations, properties, rules?
    b. What are the terms that index the referents: terminology?
4. Analysis 3
    a. What are the resources available to harvest: vocabularies, schemas, taxonomies, conceptual models, ontologies?
    b. Are there domain standards, upper/middle ontologies to embed what we create within?
5. Design 1
    a. What ontology architecture do we choose?
    b. How expressive is the ontology language we need?
    c. What conceptualization?
    d. How do we model these entities, relations, properties, rules?
    e. What are the instances of these?
    f. What data sources mappings can link to these? How?
    g. What kinds of ontology tools do we need?
6. Implement 1
    a. Implement the ontology server we will need: periodicity, granularity, configuration management
    b. Implement the infrastructure, services of our architecture: enhance the server with application, SOA support
7. Design 2
    a. Are we done with ontology development?
    b. Test competency questions as queries against ontology + data: are good answers returned quickly wrt domain experts/end users?
8. Analysis 4
    a. Refine with domain experts, end users
9. Design 3
    a. Refine conceptualization
10. Implement 2
    a. Refine ontology
11. Deploy 1
    a. Provide ontology application services
12. Deploy 2
    a. Correct problems
13. Analysis 5
    a. Interrogate users
    b. Refine requirements
    c. More resources?
14. Design 4
    a. How can changes needed be made?
    b. Refine reqs

======

More about LifeCycles:

MikeDenny

… see also: the 7 ontology lifecycle phases (from MikeDenny) as used in the OntologySummit2013 Survey of software capabilities:

- Exploration Phase
- Management Phase
- Design Phase
- Build Phase
    - General Development
    - Concept Development
    - Relationship Development
- Validation Phase
- Integration & Use Phase
- Maintenance Phase

Ref.
- slide#2 at http://ontolog.cim3.net/cgi-bin/wiki.pl?ConferenceCall_2013_02_21#nid3MTA
- and the general setup of the survey form/results - e.g. http://ontolog-02.cim3.net/wiki/Tool-123 & http://ontolog-02.cim3.net/wiki/OntologySummit2013_Survey

---

Based on MikeDenny's OntologySummit2013_Survey, as implemented by KenBaclawski on PSMW:

= Factors and Attributes of the various phase being surveyed include:

== Exploration Phase

Can Tool-123 find ontologies with specific domain coverage?
Can Tool-123 compare the domain coverage across ontologies?
Can Tool-123 assess or characterize the breadth of domain coverage of an ontology?
Can Tool-123 assess or characterize the depth or detail of domain coverage of an ontology?
Can Tool-123 assess or characterize the use considerations of ontology's licensing?
Can Tool-123 assess or characterize the use considerations of ontology's training?
Can Tool-123 assess or characterize the use considerations of ontology's cost?
Can Tool-123 assess or characterize the use considerations of ontology's updating ?

Can Tool-123 assess or characterize the use considerations of ontology's software requirements?
Can Tool-123 assess or characterize the use considerations of ontology's security?
Can Tool-123 assess or characterize other un-mentioned use considerations of an ontology?
Can Tool-123 assess the complexity of concept definitions (necessary and sufficient properties) and use of axioms?
Can Tool-123 identify number and ratio of concepts?
Can Tool-123 identify number and ratio of relations?
Can Tool-123 identify number and ratio of subclassing?
Can Tool-123 identify number and ratio of ontological constructs?

Can Tool-123 identify frequency of use of ontology language features in an ontology?
Can Tool-123 identify age and use statistics of an ontology and its versions?
Can Tool-123 assess the inferencing power of an ontology?
Can Tool-123 rate ontologies on their popularity or review feedback?

== Management Phase

Does Tool-123 distinguish between conceptual and operational ontologies and objectives?
Does Tool-123 generate ontology application and performance requirements?
Does Tool-123 organize and track the life cycle requirements and progression of an ontology?
Does Tool-123 organize and maintain a collection of related ontologies and ontology modules?
Does Tool-123 provide statistics on ontology versioning and use?
Does Tool-123 guide ontology development per a quality assurance methodology?
Can Tool-123 assess the reusability of a planned ontology?

== Design Phase

Does Tool-123 guide formulation of ontology requirements?

- Domain scope requirements
- Domain detail requirements
- Goodness of design requirements
- Data requirements
- Information requirements
- Semantic requirements
- Reasoning requirements
- Interface requirements
- Level of effort requirements

Does Tool-123 facilitate framing a set of competency questions exemplifying ontology objectives?
Does Tool-123 allow one to choose a metaphysical methodology?
Does Tool-123 allow one to choose an extant top ontology for reuse?
Does Tool-123 allow one to choose an extant core ontology for reuse?
Does Tool-123 apply a style of ontological analysis to design?

Does Tool-123 guide design to optimize normalization, factoring, and simplicity of ontology?
Does Tool-123 guide application of open world or closed world semantics?
Does Tool-123 guide ontology design to achieve inferencing requirements?
Does Tool-123 guide ontology design to achieve scalability requirements?
Does Tool-123 enable adjustable query and inference performance?
Does Tool-123 offer design patterns based on current ontology or design context?
(a) Application patterns
(b) Ontology language patterns
(c) Methodology patterns
(d) Other patterns
Does Tool-123 integrate selected design patterns into an ontology design?
Can Tool-123 assess correctness or performance of an ontology design?
Does Tool-123 guide and facilitate modularization of ontology?
Does Tool-123 guide ontology design for visualization?

== Build Phase

=== General Development

Does Tool-123 allow one to choose an ontology language?
(a) Ontology language
(b) Formal logic language
(c) RDF support?
Does Tool-123 allow one to mix ontology languages?
Does Tool-123 allow one to generate ontology code from requirements specification?
Does Tool-123 allow one to generate selected design patterns in ontology code?
Does Tool-123 allow one to enforce selected design patterns in ontology code?
Does Tool-123 provide feedback on consistency of granularity for the domain ontology structure?
Does Tool-123 provide feedback on consistency of regularity for the domain ontology structure?
Does Tool-123 manage lexical naming and annotation of ontology elements?
Does Tool-123 ensure proper use and coding of RDF and Web resources?
Does Tool-123 enforce proper use and coding of URIs?
Does Tool-123 produce the current terminological inferences of an ontology?
Does Tool-123 optimize query and inference performance?
Does Tool-123 detect violations of domain, referential, or semantic integrity?
Does Tool-123 detect overall coding errors or inconsistencies?
Does Tool-123 correct overall coding errors or inconsistencies?

=== Concept Development

Does Tool-123 ensure proper separation and coding of concepts and facts?
Does Tool-123 ensure kind-of nature and consistency of subclassing?

Does Tool-123 guide subclass versus class individual determinations?
Does Tool-123 monitor depth of subclassing consistency across a domain ontology?
Does Tool-123 generate prototypical instances (individuals) to help verify class intent
Does Tool-123 generate prototypical instances (individuals) to help verify class subsumption?
Does Tool-123 detect use of multiple inheritance?
Does Tool-123 guide use of multiple inheritance?
Does Tool-123 guide use of disjointness axioms?
Does Tool-123 guide use of necessary and sufficient properties in concept definitions?
Does Tool-123 guide use of existential quantification in class restrictions?
Does Tool-123 guide use of universal quantification in class restrictions?
Does Tool-123 guide use of existential versus universal quantification in class restrictions?
Does Tool-123 enforce proper use of conjunctions versus disjunctions?

=== Relationship Development

Does Tool-123 ensure proper use and coding of relations?
Does Tool-123 ensure proper use and coding of data types?
Does Tool-123 assess and enforce consistency and completeness of range and domain constraints?
Does Tool-123 assess and enforce consistency and completeness of inverse relations?
Does Tool-123 ensure proper use and coding of transitive relations?

== Validation Phase

Can Tool-123 verify that ontology requirements are met?
Can Tool-123 assess query performance?
(a) Query precision and recall
(b) Query time performance
Can Tool-123 assess inference performance?
(a) Inferencing time
(b) Inferencing completeness
Does Tool-123 generate characteristic queries and tests?
Does Tool-123 accept validation test sets or inputs?
Can Tool-123 assure computability of ontology?
(a) Semantically adequate
(b) Mathematically complete
Can Tool-123 verify that two ontologies are interoperable?
Can Tool-123 validate the intended functionality of software using the ontology?
Can Tool-123 validate instance data conforming to an ontology?
Can Tool-123 assess accuracy, correctness, and completeness of ontology terminological content?
Does Tool-123 guide or adjust ontology in accord with validation results?

== Integration & Use Phase

Can Tool-123 export ontology in different languages?
Can Tool-123 integrate ontology with other ontologies?
Can Tool-123 integrate ontology with other information system resources?
Can Tool-123 assess or track user experience with ontology?
Does Tool-123 enable user to modify or extend ontology to address deficiencies?
Can Tool-123 create mappings from ontologies to and from data and data sources?

== Maintenance Phase

Does Tool-123 promote reuse of ontology?
Does Tool-123 capture ontology errors during use?
Does Tool-123 profile use of ontology elements during use?
Does Tool-123 compare and map between ontologies?
Does Tool-123 track ontology changes and control versions?

- end -

DavidWhitten: “provenance (of where ideas, terms, concepts, definitions, etc come from)” may be missing in the above - see Whitten’s chat input at [+2:31]

======

Lambrix

Defects in ontologies

- Syntactic defects
    - eg. wrong tags or incorrect format
- Semantic defects
    - eg. unsatisfiable concepts, incoherent and inconsistent ontologies
- Modeling defects
    - eg. wrong or missing relations

====

Copeland
Reuse concepts from software engineering and quality assessment

- Testing at the functional requirements
    - Unit Level
    - System Level
- Testing at the non-functional requirements

- Explicit Changes

    - Asserted logical and annotation axioms
    - Properties
    - Classes
- Implicit Changes?
    - Subsumption changes
    - Entailment changes

What are Ontology Dynamics?

- Periods of growth, decline, and stability
- Axioms presence
- Types of axioms presence (e.g. continual, interrupted)
- Sequence editing types and patterns

======

Tartir - OntoQA
(from: Samir Tartir, I. Budak Arpinar, Michael Moore, Amit P. Sheth, Boanerges Aleman-meza "OntoQA: Metric-based ontology quality analysis" (2005))

Metrics (note, the following combines elements in both the presentation and the cited paper above)

- Schema Metrics
    - Relationship Richness
    - Attribute Richness
    - Inheritance Richness
    - Inheritance depth (from slides)
    - Schema depth (from slides)
- Instance Metrics
    - KB Metrics
        - Average Population
        - Cohesion (connectedness / from slides)
        - Class Utilitization (from slides)
        - Class Instance Distribution (from slides)
    - Class Metrics
        - Fullness
        - Inheritance Richness
        - Relationship Richness
        - Connectivity (centrality)
        - Readability
        - Class Importance (popularity) / from slides
        - Relationship Utilization (from slides)
    - Relationships Specific Metrics (from slides)
        - Relationship Importance (popularity)

Ontology Scoring

Score = Sum(W_i * Metric_i)
Metric_i: {Relationship diversity, Schema Depth, Class Utilization, Cohesion, Avg(Connectivity(Ci)), Avg(Importance(Ci)), Avg(Relationship Utilization(Ci)), Avg(Importance(Ri)), #Classes, #Relationships, #Instances}
W_i: Set of tunable metric weights

====
Barry Smith's ideas
Ontological realism as a strategy for integrating ontologies (Essentially - Basic Formal Ontology - BFO as an Upper Ontology) with Domain Ontology at 3rd level. ***IS THIS TOO FORMAL?***

Ontology Summit February 7, 2013 (Test this link - I was trying to link to his Presentation itself … this link works. .. but I am curious who "I" is. … now I know, it's BobSmith … Thank you, Bob. =ppy)

on REAL STUFF seems useful to include in this Requirements Doc...
Feb 7, 2013 - slide 10 -

- **Are we building ontologies so that they converge?**

(The problem of 'knowledge representation') and Strategy wrt Levels of Onto (Upper to Domain Levels?)

- **Are we using Evidence-based ontology development?**

Q: What is to serve as constraint?
A1: Authority (I tell you what to do)
A2: Homesteading (Founder effect)
A3: Best candidate terminology
But what does 'best' mean?
A4: Voting ?
But then on what grounds should people vote?

Page 15 -

- • Level 1: the entities in reality, both instances and universals

• Level 2: cognitive representations of this reality, e.g. on the part of scientists ...

- • Level 3: publicly accessible concretizations of these cognitive representations in textual and graphical artifacts
- **Are we keeping the Three Levels "Straight"?**

Page 17 http://iospress.metapress.com/content/1551884412214u67/?issue=3&genre=article&spage=139&issn=1570-5838&volume=5
Smith and Ceusters, “Ontological Realism as a Methodology for Coordinated Evolution of Scientific Ontologies”, Applied Ontology, 5 (2010), 139–188

====

Duque Ramos et al - OQuare

ref: http://miuras.inf.um.es/evaluation/oquare/
ref2: http://miuras.inf.um.es/oquarewiki
also: Duque-Ramos et al - “OQuaRE: A SQuaRE-based approach for evaluating the quality of ontologies” (2011)

Characteristics

- **Structural:** Formal and semantic important ontological properties that are widely used in state of-the-art evaluation approaches. Some sub-characteristics are formalisation, formal relations support, cohesion, tangledness, redundancy and consistency.
- **Functional adequacy**: An ontology is evaluated for this criterion according to the degree of accomplishment of functional requirements, that is, the appropriateness for its intended purpose according to Stevens and Lord (2009): reference ontology, controlled vocabulary, schema and value reconciliation, consistent search and query, knowledge acquisition, clustering and similarity, indexing and linking, results representation, classifying instances, text analysis, guidance and decision trees, knowledge reuse, inferencing, and precision.
- **Reliability**: Capability of an ontology to maintain its level of performance under stated conditions for a given period of time. Recoverability and availability are some of its sub-characteristics.
- **Operability**: Effort needed for using an ontology, and in the individual assessment of such use, by a stated or implied set of users, and it is measured through sub-characteristics such as learnability.
- **Maintainability**: The capability of ontologies to be modified for changes in environments, in requirements or in functional specifications. Some sub-characteristics are modularity, reusability, analysability, changeability, modification stability and testability.
- **Compatibility:** The ability of two or more software components to exchange information and/or to perform their required functions while sharing the same hardware or software environment
- **Transferability**: The degree to which the software product can be transferred from one environment to another
- **Performance efficiency:** Relationship between the level of performance of the software and the amount of resources used, under stated conditions, taking into account elements such as the time response, or memory consumption.

- **Quality in use:** Quality in a particular context of use. Quality in use is the degree to which a product used by specific users meets their needs to achieve specific goals

Ontology Quality Metrics

Sub-characteristics

Structural

- Formalisation: An efficient ontology has to be built on top of a formal model to support reasoning.
- Formal relations support: Most ontologies only have formal support for taxonomy. The usage of additional formal theories would be a positive indicator.
- Cohesion: An ontology has a high cohesion if the classes are strongly related.
- Tangledness: This measures the distribution of multiple parent categories, so that it is related to the existence of multiple inheritance, which is usually a sign of suboptimal design.
- Redundancy: Capability of the ontology to be informative
- Structural accuracy: Degree of the correctness of the terms used in the ontology
- Consistency: Degree of the consistency of the ontology
- Cycles: The existence of cycles through a particular semantic relation is usually a sign of bad design, since they may lead to inconsistencies.

Functional adequacy

- Schema and value reconciliation: An ontology can provide a common data model that can be applied to particular views for their reconciliation and integration. Ontologies facilitate the achievement of semantic interoperability if they are able to provide the semantic context for data and information.
- Consistent search and query:
    - The formal model of the ontology allows for better querying and searching methods.
    - The ontology structure can guide search processes if they provide a semantic context to evaluate the data wanted by the users.
    - This semantic context is provided by the concepts,
    - This semantic context is provided by all the machine computable properties
    - This semantic context is provided by the axioms.
- Knowledge reuse: Degree to which the knowledge of an ontology can be used to build other ontologies.
- Knowledge acquisition: Ontologies are templates for generating the forms by which instances are acquired.
- Domain coverage: The degree to which the ontology covers the specified domain
- Reference ontology: Degree in which the ontology can be used as a reference resource for the particular domain the ontology is built for.
- Controlled vocabulary: Capability of the ontology to avoid heterogeneity of the terms
- Knowledge acquisition – representation.: Capability of the Ontology to represent the knowledge acquired.

- Clustering: Degree in which the annotations of data with respect to ontology terms can be used for clustering such data against the aspects of the ontology
- Similarity: Capability of the component of the ontology to be compared for similarity
- Indexing and linking: Degree in which the classes defined in the ontology can act as indexes for quick information retrieval
- Results representation: Capability of the ontology to analyze complex results such as microarrays experiments
- Classifying instances: Degree in which ontology Instances can be recognized as member of a certain class
- Text analysis: Capability of the structure of the ontology to help detect associations between words or concepts and classifying word types.
- Guidance: Capability of the ontology to guide the specification of domain theories.
- Decision trees: Capability of the ontology to be used building Decision trees.
- Knowledge reuse: The degree to which the ontology knowledge can be used to build other ontologies.
- Inference: The degree to which the formal model of the ontology can be used by reasoners to make implicit knowledge explicit.

Maintainability

- Modularity: The degree to which the ontology is composed of discrete components such that a change to one component has a minimal impact on other components. Changes in ontologies may lead to unexpected effects, like inconsistencies, in the ontology.
- Reusability: The degree to which a part of the ontology can be reused in more than one ontology, or to build other ontologies.
- Analysability: The degree to which the ontology can be diagnosed for deficiencies or causes of inconsistencies, or for the parts to be modified to be identified.
- Changeability: The degree to which the ontology enables a specified modification to be implemented. The ease with which an ontology can be modified
- Modification stability: The degree to which the ontology can avoid unexpected effects from modifications of the knowledge (terms, classes, properties, etc..).
- Testability: The degree to which the modified ontology can be validated.

Compatibility

- Replaceability: The degree to which the ontology can be used in place of another specified Ontology for the same purpose in the same environment.
- Interoperability: The degree to which the ontology can be cooperatively operable combining its knowledge with one or more other ontologies.

Transferability

- Portability: The degree in which an Ontology or one part of the ontology can be transferred from one hardware or software environment to another
- Adaptability: The degree to which The ontology can be adapted for different specified environments (languages, expressivity levels) without applying actions or means other than those provided for this purpose for the Ontology considered.

Operability

- Appropriateness recognisability: The degree to which the Ontology enables users to recognise whether it is appropriate for their needs.
- Learnability: The degree to which the ontology enables users to learn its application.
- Ease of operation: The degree to which the ontology makes it easy for users to operate it
- Ease of control: The degree to which the ontology makes it easy for users to control it.
- Ease of use: The degree to which the ontology makes it easy for users to use it in some other way.
- Helpfulness: The degree to which the ontology provides help when users need assistance.

Reliability

- Error detection: The degree to which the ontology enables users to detect faults.
- Recoverability: The degree to which the ontology can re-establish a specified level of performance and recover the data directly affected in the case of a failure.
- Availability: The degree to which an ontology or part of it is operational and available when required for use with different applications

Performance Efficiency

- Response time: The degree to which the ontology provides appropriate response times from performing its function (Queries and reasoning) under stated conditions
- Processing time: The degree to which the ontology provides appropriate processing times from performing its function (Queries and reasoning) under stated conditions
- Throughput rates: The degree to which the ontology provides appropriate throughput rates when performing its function (Queries and reasoning), under stated conditions.
- Resource Utilization: The degree to which the application uses appropriate amounts of resources when the ontology performs its function (Queries and reasoning) under stated conditions.
- Resource Utilization: The degree to which the application uses appropriate types of resources when the ontology performs its function (Queries and reasoning) under stated conditions.

Metrics (page 47 of the paper)

• Lack of Cohesion in Methods (LCOMOnto):

The semantic and conceptual relatedness of classes can be used to measure the separation of responsibilities and independence of components of ontologies.

It is calculated as follows:

$LCOMOnto=\sum path(|C(leaf)i|)/m$,

where path $|C(leaf)i|$ is the length of the path from the leaf class i to Thing,

where m is the total number of paths in the ontology.

• Weighted Method Count (WMCOnto):

Mean number of properties and relationships per class.

It is calculated as follows:

$WMCOnto=(\sum|P\ Ci|+\sum|\ RCi|)/\sum|Ci|$,

where Ci is the i-th class in the ontology.

• Depth of subsumption hierarchy (DITOnto):

Length of the largest path from Thing to a leaf class.
It is calculated as follows:
DITOnto=Max (∑D|Ci |),
where Ci are the classes
where D|Ci | is length of the path from the ith leaf class of the ontology to Thing.

• Number of Ancestor Classes (NACOnto):
Mean number of ancestor classes per leaf class.
It is the number of direct superclasses per leaf class,
It is calculated as follows:
NACOnto=∑|SupC(Leaf)i|/∑|C(leaf)i)|

• Number of Children (NOCOnto):
Mean number of direct subclasses.
It is the number of relationships divided by the number of classes minus the relationships of Thing)
It is calculated as follows:
NOCOnto=∑| R Ci|/(∑|Ci | -| RThing|)

• Coupling between Objects (CBOOnto):
Number of related classes.
It is the average number of the direct parents per class minus the relationships of Thing,
It is calculated as follows:
CBOOnto=∑|SupCi|/(∑|Ci | -| RThing|)

• Response for a class (RFCOnto):
Number of properties that can be directly accessed from the class.
It is calculated as follows:
RFCOnto=(∑|P Ci|+∑|SupCi|/(∑|Ci| -| RThing|)

• Number of properties (NOMOnto):
Number of properties per class.
It is calculated as follows:
NOMOnto=∑| PCi|/∑|Ci |

• Properties Richness (RROnto):
Number of properties defined in the ontology divided by the number of relationships and properties.
It is calculated as follows:
RROnto=∑| P Ci|/∑(| R Ci| + ∑|Ci|)

• Attribute Richness (AROnto):
Mean number of attributes per class.
It is calculated as follows:
AROnto=∑|AttCi|/∑|Ci |

• Relationships per class (INROnto):

Mean number of relationships per class.
It is calculated as follows:
INROnto=∑| R Ci| / ∑|Ci |

• Class Richness (CROnto):
Mean number of instances per class.
It is calculated as follows:
CROnto=∑| I Ci| / ∑|Ci |;
where I Ci, is the set of individuals of the Ci class.

• Annotation Richness (ANOnto):
Mean number of annotations per class.
It is calculated as follows:
ANOnto=∑| A Ci| / ∑|Ci |;
where Ci is the i-th class in the ontology.

• Tangledness (TMOnto):
Mean number of parents per class.
It is calculated as follows:
TMOnto=∑| R Ci| / ∑|Ci |-∑|C(DP)i |;
where Ci is the i-th class in the ontology
where C(DP)i is the i-th class in the ontology with more than one direct parent.

==========

Ontology Features:

- Annotation
- Argument Constraints
- Inter-Argument Constraints
- Argument Type Conclusion
- Denotational Functions
- Metaclasses
- Non-Binary Predicates
- Single Class Hierarchy Root
- Multiple Class Hierarchy Roots
- Class Covering
- Class Partitioning
- Term Mapping To Natural Language
- Term Mapping To Controlled Vocabulary
- Ontology Inheritance
- Open world reasoning
- Closed world reasoning

Logic Features (determine class of logic: type of DL, First-Order, Second-Order, …)

- Atomic Negation
- Concept Intersection
- Complex Concept Negation
- Universal Quantification
- Full Existential Quantification
- Transitive Roles
- Inverse Properties
- Irreflexivity
- Reflexivity
- Qualified Cardinality Restrictions
- Binary Predicate Disjointness
- First Order Implications
- Quantification Over Classes
- Quantification Over Predicates
- Quantification Over Functions
- Enumerated Classes
- Datatypes

Ontology Tool features

- Reasoning upon data entry - Detection of self-disjointness
- Reasoning upon data entry - Detection of predicate inheriting from two incompatible predicates

====
; Cyc ontology of some ontology evaluation concepts

default mt: UniversalVocabularyMt.
Constant: OntologyEvaluationVocabularyMt.
isa: VocabularyMicrotheory.
genlMt: UniversalVocabularyMt.
in mt: OntologyEvaluationVocabularyMt.
comment: "The ontology for terms for expressing characteristics of ontologies, ontology languages, and evaluation of both.".

Constant: OntologyEvaluationMt.
isa: TheoryMicrotheory.
genlMt-Vocabulary: OntologyEvaluationVocabularyMt.
in mt: OntologyEvaluationMt.
comment: "The ontology for expressing characteristics of ontologies, ontology languages, and evaluation of both.".

default mt: OntologyEvaluationVocabularyMt.
Constant: Metaclass.
isa: VariedOrderCollection CollectionType.
genls: CollectionType.
comment: "A #$Metaclass is a class all of whose instances are themselves classes.".
definingMt: OntologyEvaluationVocabularyMt.

; ==========

Constant: TemporalClass.
isa: Metaclass SecondOrderCollection.
genls: FirstOrderCollection.
typeGenls: TemporalThing.
comment: "A #$TemporalClass is a class which is a subclass of #$TemporalThing.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: RigidClass.
isa: Metaclass SecondOrderCollection.
genls: FirstOrderCollection.
typeGenls: Individual.
comment: "A #$RigidClass is a class, all of whose instances are individuals which are necessarily instances of the class so long as they exist. Instances of #$RigidClass may exist in time (e.g. Homo sapiens) or be timeless (e.g., numbers or Platonic shapes).".
definingMt: OntologyEvaluationVocabularyMt.
in mt: OntologyEvaluationMt.
f: (implies

(and
(isa ?RCLASS RigidClass)
(holdsIn ?TIME1 (isa ?X ?RCLASS))
(temporallySubsumes ?TIME2 ?RCLASS))
(holdsIn ?TIME2 (isa ?X ?RCLASS))).

Constant: RigidTemporalClass.
isa: Metaclass SecondOrderCollection.
genls: RigidClass TemporalClass.
comment: "A #$RigidTemporalClass is a temporal class, all of whose instances are individuals which are necessarily instances of the class so long as they exist.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: SemiRigidClass.
isa: Metaclass SecondOrderCollection.
genls: TemporalClass.
disjointWith: RigidClass.
comment: "A #$SemiRigidClass is a temporal class, some of whose instances are individuals which are necessarily instances of the class so long as they exist, while other instances may cease to be instances of the class yet continue to exist. E.g., HardObject, since diamonds are necessarily hard, but the hardness of a piece of plastic may depend upon its temperature.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: AntiRigidClass.
isa: Metaclass SecondOrderCollection.
genls: TemporalClass.
disjointWith: RigidClass SemiRigidClass.
comment: "An #$AntiRigidClass is a temporal class, none of whose instances are necessarily instances of the class so long as they exist. E.g., Baby or Student.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: InitialClass.
isa: Metaclass SecondOrderCollection.
genls: AntiRigidClass.
comment: "An #$InitialClass is a temporal class, all of whose instances are individuals which are necessarily instances of the class when the instance started to exist, but may cease to be instances of the class yet continue to exist. Once they cease to be instances of the class, they can not later become instances again. E.g., Virgin.".
definingMt: OntologyEvaluationVocabularyMt.
in mt: OntologyEvaluationMt.

f: (implies
(and

(isa ?ICLASS InitialClass)
(holdsIn ?TIME1 (isa ?X ?ICLASS))
(startingPoint ?X ?TIME2))
(holdsIn ?TIME2 (isa ?X ?ICLASS))).
f: (implies
(and
(isa ?ICLASS InitialClass)
(holdsIn ?TIME1 (isa ?X ?ICLASS))
(endsAfterEndingOf ?TIME1 ?TIME2)
(startsAfterStartingOf ?TIME2 ?X))
(holdsIn ?TIME2 (isa ?X ?ICLASS))).

Constant: NonInitialClass.
isa: Metaclass SecondOrderCollection.
genls: AntiRigidClass.
disjointWith: InitialClass.
comment: "A #$NonInitialClass is a temporal class, none of whose instances were instances of the class when the instance started to exist, but later became instances of the class. E.g., Student.".
definingMt: OntologyEvaluationVocabularyMt.
in mt: OntologyEvaluationMt.
f: (implies
(and
(isa ?NICLASS NonInitialClass)
(holdsIn ?TIME1 (isa ?X ?NICLASS))
(startingPoint ?X ?TIME2))
(not (holdsIn ?TIME2 (isa ?X ?ICLASS)))).

Constant: TerminalClass.
isa: Metaclass SecondOrderCollection.
genls: AntiRigidClass.
disjointWith: InitialClass.
comment: "A #$TerminalClass is a temporal class, some of whose instances were not instances of the class when the instance started to exist, but once they became instances of the class they continue to be instances while they continue to exist. E.g., Murderer.".
definingMt: OntologyEvaluationVocabularyMt.
in mt: OntologyEvaluationMt.

f: (implies
(and
(isa ?TCLASS TerminalClass)

```
        (holdsIn ?TIME1 (isa ?X ?TCLASS))
        (endingPoint ?X ?TIME2))
        (holdsIn ?TIME2 (isa ?X ?TCLASS))).
f: (implies
        (and
        (isa ?TCLASS TerminalClass)
        (holdsIn ?TIME1 (isa ?X ?TCLASS))
        (startsAfterStartingOf ?TIME2 ?TIME1)
        (endsAfterEndingOf ?X ?TIME2))
        (holdsIn ?TIME2 (isa ?X ?TCLASS))).

Constant: OccasionalClass.
isa: Metaclass SecondOrderCollection.
genls: NonInitialClass.
disjointWith: TerminalClass.
comment: "An #$OccasionalClass is a temporal class, whose instances may be instances of
the class at one time and not at another.  An individual after it has ceased being an instance of
#$OccasionalClass can become one again.  E.g., Student.".
definingMt: OntologyEvaluationVocabularyMt.

; ============================================================

Constant: ClassificationSystem.
isa: ObjectType.
genls: AspatialInformationStore.
comment: "A specialization of #$AspatialInformationStore.  Each instance of
#$ClassificationSystem is an agreed-upon or conventional classification system consisting of
several #$ConventionalClassificationTypes.  In such a system, a change or reclassification is
possible by a decision of an authority, or by a changed social agreement, without changing the
intrinsic natures of the actual objects in the classificatory categories.  ClassificationSystems
include biological taxonomies, standard classifications, data dictionaries, thesauri, and calendar
systems.".

Constant: termInClassificationSystem.
isa: BinaryPredicate.
quotedArgument: 1.
arg1QuotedIsa: Thing.
arg2Isa: ClassifiationSystem.


comment: "<code>(#$termInClassificationSystem TERM CLASSIFICATION)</code> means
that the term TERM is classified in the #$ClassificationSystem <code>CLASSIFICATION</
code>.".
```

Constant: Taxonomy.
isa: ObjectType.
genls: ClassificationSystem.
comment: "The class of all taxonomic systems. A taxonomy of classes includes a hierarchical division of a domain along class/subclass (see #$ClassSubclassHierarchy) lines, partonomic lines, or possibly similar transitive binary predicates (e.g. derivedFrom). A taxonomy of relations may be specified using subrelationOf as the taxonomic predicate. A #$StrictTaxonomy is one in which each node has a single parent and all siblings are disjoint. A taxonomy can be usefully contrasted with an ontology (see Ontology), the latter of which includes a rich description of a domain, often including non-binary relations and complex rules.".

Constant: taxonomicRelationOf.
isa: BinaryPredicate.
arg1Isa: Taxonomy.
arg2Isa: BinaryPredicate.
comment: "(#$taxonomicRelationOf BIN_PRED TAXONOMY) means that the binary relation BIN_PRED is the taxonomic relation for the #$Taxonomy TAXONOMY.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: StrictTaxonomy.
isa: ObjectType.
genls: Taxonomy.
comment: "The class of all taxonomic systems in which each node has a single parent and all siblings are disjoint.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: ClassSubclassHierarchy.
isa: ObjectType.
genls: Taxonomy.
comment: "The collection of all taxonomic systems that describe a domain only along class (elementOf) and sub-class (subsetOf) lines. Such a hierarchy may be either a #$StrictTaxonomy or a #$SubclassDAG.".

Constant: SubclassDAG.
isa: ObjectType.
genls: ClassSubclassHierarchy.
disjointWith: StrictTaxonomy.
comment: "The collection of all taxonomic systems that describe a domain only along class (elementOf) and sub-class (subsetOf) lines and permit classes to have multiple superclasses.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: StrictClassSubclassTaxonomy.
isa: ObjectType.
genls: ClassSubclassHierarchy StrictTaxonomy.

comment: "The collection of all taxonomic systems that describe a domain only along class (elementOf) and sub-class (subsetOf) lines. Such a hierarchy may be either a #$StrictTaxonomy or a #$SubclassDAG.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: PartonomicTaxonomy.
isa: ObjectType.
genls: Taxonomy.
disjointWith: ClassSubclassHierarchy.
comment: "The collection of all taxonomic systems that describe a domain only along partonomic (partTypeOf) lines. Such a hierarchy may be either a #$StrictTaxonomy or a #$SubclassDAG. The partonomic relation may be, for example, temporal, physical, or organizational. The relation may be narrower in one part of the taxonomy than in another.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: partonomicRelationForTaxonomy.
isa: BinaryPredicate.
arg1Isa: PartonomicTaxonomy.
arg2Isa: BinaryPredicate.
genlPreds: taxonomicRelationOf.
comment: "(#$partonomicRelationForTaxonomy PART_PRED PART_TAXONOMY) means that the partonomic relation (i.e., a subrelation of #$partTypes) PART_PRED is the taxonomic relation for the #$PartonomicTaxonomy PART_TAXONOMY.".
definingMt: OntologyEvaluationVocabularyMt.

Constant: Ontology.
isa: ObjectType.
genls: ClassificationSystem Technology-Artifact StructuredInformationSource .
comment: "The collection of organized conceptual structures in which types of things form a generalization hierarchy and in which properties are defined to hold between instances of the types of things.".
in mt: OntologyEvaluationMt.
properPartTypes: ClassSubclassHierarchy.
f: (relationAllExists properSubSystems Ontology ClassSubclassHierarchy).

; ==

Constant: Relation.
isa : RelationshipType.
genls : AbstractIndividual MathematicalObject.

comment : "#$Relation is the class of all relations. Each instance of Relation is a relation that can hold among one or more things, depending on whether the relation is unary, binary, ternary, ... (see #$arity and #$relationalArity).

An important subcollection of #$Relation is #$Predicate (q.v.), whose instances return true or false (with respect to a given context or interpretation).

Another important subcollection of #$Relation is #$Function-Denotational (q.v.), the collection of all functions.".

Constant: FixedArityRelation.
isa : RelationshipType.
genls : Relation.
comment: "#$FixedArityRelation is the collection of all relations whose arity is constant, so that the relation #$arity applies.".

Constant: arity.
isa: BinaryPredicate.
arg1Isa: FixedArityRelation.
arg2Isa: NonNegativeInteger.
comment: "A #$Relation used for stating that a given relation takes a specified number of arguments. <code>(#$arity RELN N)</code> means that that semantic well-formedness requires that <code>RELN</code> take exactly <code>N</code>arguments at a time. That is, a formula <code>(RELN ARG1 ... ARGM)</code> is semantically well-formed only if <code>M = N</code>.

For example, the arity of any instance of #$BinaryPredicate is 2.".

Constant: UnaryRelation.
isa : RelationshipType.
genls : FixedArityRelation.
comment: "#$UnaryRelation is the collection of all relations whose arity is 1.".
f: (relationAllInstance arity UnaryRelation 1).

Constant: BinaryRelation.
isa : RelationshipType.
genls : FixedArityRelation.
comment: "#$BinaryRelation is the collection of all relations whose arity is 2.".
disjointWith: UnaryRelation.
f: (relationAllInstance arity BinaryRelation 2).

Constant: TernaryRelation.
isa : RelationshipType.

genls : FixedArityRelation.
comment: "#$TernaryRelation is the collection of all relations whose arity is 3.".
disjointWith: UnaryRelation BinaryRelation.
f: (relationAllInstance arity TernaryRelation 3).

Constant: QuarternaryRelation.
isa : RelationshipType.
genls : FixedArityRelation.
comment: "#$QuarternaryRelation is the collection of all relations whose arity is 4.".
disjointWith: UnaryRelation BinaryRelation TernaryRelation.
f: (relationAllInstance arity QuarternaryRelation 4).

Constant: VariableArityRelation.
isa : RelationshipType.
genls : Relation.
comment: "#$VariableArityRelation is the collection of all relations whose arity is not restricted to a single value. For example, a function or predicate for expressing sums of two or more values.".
disjointWith: FixedArityRelation.

Constant: Predicate.
isa : RelationshipType.
genls : Relation.
comment : "#$Predicate is the class of all relations whose instances return true or false (with respect to a given context or interpretation). Each instance of #$Predicate is either a property of things (see #$UnaryPredicate) or a relationship holding between two or more things. Expressions using predicates along with their arguments are used to form sentences.".

Constant: FixedArityPredicate.
isa : RelationshipType.
genls : Predicate FixedArityRelation.
comment: "#$FixedArityPredicate is the collection of all predicates whose arity is constant, so that the relation #$arity applies.".

Constant: UnaryPredicate.
isa : RelationshipType.
genls : FixedArityPredicate UnaryRelation.
comment: "#$UnaryPredicate is the collection of all predicates whose arity is 1.".

Constant: BinaryPredicate.

isa : RelationshipType.
genls : FixedArityPredicate BinaryRelation.
comment: "#$BinaryPredicate is the collection of all predicates whose arity is 2.".

Constant: TernaryPredicate.
isa : RelationshipType.
genls : FixedArityPredicate TernaryRelation.
comment: "#$TernaryPredicate is the collection of all predicates whose arity is 3.".

Constant: QuarternaryPredicate.
isa : RelationshipType.
genls : FixedArityPredicate QuarternaryRelation.
comment: "#$QuarternaryPredicate is the collection of all predicates whose arity is 4.".

Constant: VariableArityPredicate.
isa : RelationshipType.
genls : Predicate VariableArityRelation.
comment: "#$VariableArityPredicate is the collection of all predicates whose arity is not restricted to a single value. For example, a predicate for expressing sums of two or more values.".

Constant:ReflexiveBinaryPredicate.
isa : RelationshipType.
genls : BinaryPredicate .
comment: "A #$BinaryPredicate PRED is an #$ReflexiveBinaryPredicate if and only if it has the following two properties. (i) For any THING1 that meets the argument constraints of each of PRED's argument places (PRED THING1 THING1) holds. (ii) PRED's two argument-places must be co-satisfiable with respect to their type constraints.".
f: (implies
	(and
	(isa ?PRED ReflexiveBinaryPredicate)
	(arg1Isa ?PRED ?ARG1TYPE)
	(arg2Isa ?PRED ?ARG2TYPE)
	(genls ?ARG1TYPE Individual)
	(isa ?X ?ARG1TYPE)
	(isa ?X ?ARG2TYPE))
	(?PRED ?X ?X)).
f: (implies
	(and
	(isa ?PRED ReflexiveBinaryPredicate)
	(arg1Isa ?PRED ?ARG1TYPE)
	(arg2Isa ?PRED ?ARG2TYPE)
	(arg1Genl ?PRED ?ARG1SUPER)
	(arg2Genl ?PRED ?ARG2SUPER))

(isa ?X ?ARG1TYPE)
(isa ?X ?ARG2TYPE)
(genl ?X ?ARG1SUPER)
(genl ?X ?ARG2SUPER))
(?PRED ?X ?X)).

Constant: IrreflexiveBinaryPredicate.
isa : RelationshipType.
genls : BinaryPredicate .
comment: "A #$BinaryPredicate PRED is an #$IrreflexiveBinaryPredicate if and only if it has the following two properties. (i) For any THING1 that meets the argument constraints of each of PRED's argument places (PRED THING1 THING1) does not hold. (ii) PRED's two argument-places must be co-satisfiable with respect to their type constraints.".
disjointWith: ReflexiveBinaryPredicate.
f: (implies
(and
(isa ?PRED IrreflexiveBinaryPredicate)
(?PRED ?X ?Y))
(different ?X ?Y)).

Constant: SymmetricBinaryPredicate.
isa : RelationshipType.
genls : BinaryPredicate.
comment: "A #$BinaryPredicate PRED is a #$SymmetricBinaryPredicate if and only if for any things X and Y, if (PRED X Y) then (PRED Y X). The argument type constraints on PRED's two argument-places are co-satisfiable.".
f: (implies
(and
(isa ?PRED SymmetricBinaryPredicate)
(?PRED ?X ?Y))
(?PRED ?Y ?X)).

Constant: AntiSymmetricBinaryPredicate.
isa : RelationshipType.
genls : BinaryPredicate.
comment: "A #$BinaryPredicate PRED is an #$AntiSymmetricBinaryPredicate if and only if it has the following two properties. (i) For any THING1 and THING2 such that both (PRED THING1 THING2) and (PRED THING2 THING1) hold, THING1 = THING2. (ii) PRED's two argument-places must be co-satisfiable with respect to their type constraints.".
f: (implies
(and
(isa ?PRED AntiSymmetricBinaryPredicate)
(?PRED ?X ?Y)
(?PRED ?Y ?X))

(equals ?X ?Y)).

Constant: AsymmetricBinaryPredicate.
isa : RelationshipType.
genls : AntiSymmetricBinaryPredicate IrreflexiveBinaryPredicate.
comment: "A #$BinaryPredicate PRED is an #$AsymmetricBinaryPredicate if and only if it has the following two properties. (i) For any THING1 and THING2 such that if (PRED THING1 THING2) holds, (PRED THING2 THING1) does not hold. (ii) PRED's two argument-places must be co-satisfiable with respect to their type constraints.".
f: (implies
(and
(isa ?PRED AsymmetricBinaryPredicate)
(?PRED ?X ?Y))
(not (?PRED ?Y ?X))).

Constant: CoEquivalenceBinaryPredicate.
isa : RelationshipType.
genls : SymmetricBinaryPredicate IrreflexiveBinaryPredicate.
comment: "A #$BinaryPredicate PRED is an #$CoEquivalenceBinaryPredicate if and only if it has the following two properties. (i) For any THING1 and THING2 such that if (PRED THING1 THING2) holds and THING1 is different from THING2, (PRED THING2 THING1) holds. (ii) Thre is no THING1 such that (PRED THING1 THING1) holds. (iii) PRED's two argument-places must be co-satisfiable with respect to their type constraints.".

Constant: ReflexiveBinaryPredicate.
isa : RelationshipType.
genls : BinaryPredicate.
comment: "A #$ReflexiveBinaryPredicate is a #$BinaryPredicate whose instances are reflexive relations. A reflexive binary predicate relates anything that meets the predicate's argument constraints to that thing itself. A reflexive predicate has precisely the same constraints on both of its argument-places.

More precisely: a binary predicate PRED is an instance of ReflexiveBinaryPredicate if and only if both (i) for every X that satisfies the constraints on (i.e is an admittedArgument for) both argument-places of PRED, (PRED X X) holds and (ii) PRED's two argument-places are co-satisfiable with respect to their (single-argument) type constraints.".

Constant: TransitiveBinaryPredicate .
isa : RelationshipType.
genls : BinaryPredicate.

comment: "A specialization of #$BinaryPredicate. A binary predicate PRED is an instance of #$TransitiveBinaryPredicate only if it has the following property: For any THING1, THING2, and THING3, if both of these hold:

(PRED THING1 THING2)
(PRED THING2 THING3)

Then so does this:

(PRED THING1 THING3) .

PRED's two argument-places must be co-satisfiable.".

Constant: AntiTransitiveBinaryPredicate.
isa : RelationshipType.
genls : IrreflexiveBinaryPredicate.
comment: "A specialization of #$BinaryPredicate. A binary predicate PRED is an instance of #$AntiTransitiveBinaryPredicate only if it has the following property: For any THING1, THING2, and THING3, if both of these hold:

(PRED THING1 THING2)
(PRED THING2 THING3)

Then :
(PRED THING1 THING3)
does not hold.

PRED's two argument-places must be co-satisfiable.".

Constant: FunctionalPredicate.
isa : RelationshipType.
genls : Predicate UnaryRelation.
comment: "#$FunctionalPredicate is the collection of all predicates such that if a statement headed by that predicate is true with all but (one of) its functional argument bound, there is no true statement with that argument substituted by any non-equal value.".
disjointWith: CoEquivalenceBinaryPredicate.

f: (implies
(and
(isa ?FPRED FunctionalPredicate)
(functionalInArg ?FPRED 1)
(?FPRED ?X)
(?FPRED ?Y))

```
        (equals ?X ?Y)).
f: (implies
        (and
        (isa ?FPRED FunctionalPredicate)
        (functionalInArg ?FPRED 1)
        (?FPRED ?X ?A2)
        (?FPRED ?Y ?A2))
        (equals ?X ?Y)).
f: (implies
        (and
        (isa ?FPRED FunctionalPredicate)
        (functionalInArg ?FPRED 1)
        (?FPRED ?A1 ?X)
        (?FPRED ?A1 ?Y))
        (equals ?X ?Y)).
; and similar rules for higher arity
f: (implies
        (and
        (isa ?PRED ReflexiveBinaryPredicate)
        (isa ?PRED FunctionalPredicate))
        (isa ?PRED AntiSymmetricBinaryPredicate)).
```

Constant: Function-Denotational.
isa : RelationshipType.
genls : Relation.
disjointWith : Predicate
comment : "A specialization of #$Relation (q.v.) and the collection of all functions. Each instance of #$Function-Denotational is a many-one relation that represents, with respect to any given context, a mapping from one set of things (the function's relationDomain) to another set of things (its relationRange). As a many-one relation, a function maps each thing in its domain to exactly one thing in its range. A function might be unary, binary, ternary, etc. depending on whether the members of its domain are singletons, ordered pairs, ordered triples, etc. The function is said to take the items from any its N arguments and to return a member of its range as its corresponding value . There are also functions that are not of any particular fixed arity; see #$VariableArityFunction.

Note that the domain and range (i.e. the extension) of a given function might vary from context to context. For example, as the government of a country can change over time, GovernmentFn will in such cases map the same country to different governments with respect to different times. Thus, an instance of Function-Denotational is closer to what is sometimes called a ``function-in-intension'' than to the purely set-theoretical notion of a ``function-in-extension''.

Function-Denotational does not include any of the so-called truth functions: Predicates, Quantifiers, or LogicalConnectives." .

; ==========

Constant: OntologyWithArgumentConstraints.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies in which type constraints on arguments for relations can be expressed.".

Constant: OntologyWithoutArgumentConstraints.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies in which type constraints on arguments for relations cannot be expressed.".
disjointWith: OntologyWithArgumentConstraints.

Constant: OntologyWithInterArgumentConstraints.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies in which type constraints between arguments for relations can be expressed.".

Constant: OntologyWithoutInterArgumentConstraints.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies in which type constraints between arguments for relations cannot be expressed.".
disjointWith: OntologyWithArgumentConstraints.

Constant: OntologyWithArgumentTypeConclusion.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies in which the class of an argument may be logically concluded from its use in a relation.".

Constant: OntologyWithoutArgumentTypeConclusion.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies in which the class of an argument may not be logically concluded from its use in a relation.".

```
disjointWith: OntologyWithArgumentTypeConclusion.

Constant: OntologyWithDenotationalFunctions.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies which contains denotational functions (instances of
#$Function-Denotational).".
f: (implies
        (isa ?ONTOL OntologyWithDenotationalFunctions)
        (thereExists ?FUNC
        (and
        (isa ?FUNC Function-Denotational)
        (termInClassificationSystem ?FUNC ?ONTOL)))).
f: (implies
        (and
        (isa ?FUNC Function-Denotational)
        (termInClassificationSystem ?FUNC ?ONTOL))
        (isa ?ONTOL OntologyWithDenotationalFunctions)).

Constant: OntologyWithoutDenotationalFunctions.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies which contains denotational functions (instances of
#$Function-Denotational).".
disjointWith: OntologyWithDenotationalFunctions.

Constant: OntologyWithOnlyBinaryPredicates.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies all of whose predicates are binary.".
f: (implies
        (and
        (isa ?PRED Predicate)
        (isa ?ONTOL OntologyWithOnlyBinaryPredicates)
        (termInClassificationSystem ?PRED ?ONTOL))
        (isa ?PRED BinaryPredicate)).

Constant: OntologyWithNonBinaryPredicates.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies which contains predicates that are not binary.".
disjointWith: OntologyWithOnlyBinaryPredicates.
f: (implies
        (and
```

(isa ?PRED Predicate)
(not (isa ?PRED Predicate))
(termInClassificationSystem ?PRED ?ONTOL))
(isa ?ONTOL OntologyWithNonBinaryPredicates)).

Constant: OntologyWithSingleClassHierarchyRoot.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies all of whose classes share a single common superclass.".

Constant: OntologyWithoutSingleClassHierarchyRoot.
isa: ObjectType.
genls: Ontology.
comment: "The collection of ontologies which have classes that do not have a common superclass.".
disjointWith: OntologyWithSingleClassHierarchyRoot.

Constant: ClassCovering.
isa: OntologyFeature.
comment: "Class covering is the definition of covering sets of subclasses of a class.".

Constant: ClassPartitioning.
isa: OntologyFeature.
genls: ClassCovering.
comment: "Class partitioning is the definition of disjoint sets of covering subclasses of a class.".

Constant: TermMappingToNL.
isa: OntologyFeature.

Constant: TermMappingToControlledVocabulary.
isa: OntologyFeature.

Constant: OntologyInheritance.
isa: OntologyFeature.

; ==

Constant: LogicFeature.
isa: ObjectType.
genls: RelationalStructure.
comment: "".

Concept: AtomicNegation.
isa: LogicFeature.
comment: "#$AtomicNegation is negation of concept names that do not appear on the consequent of axioms.".
termStrings: "atomic negation".

Concept: ClassIntersection.
isa: LogicFeature.
comment: "#$ClassIntersection is the logical feature of computing the intersection of two classes/types of things.".
termStrings: "class intersection" "concept intersection".

Concept: UniversalQuantification.
isa: LogicFeature.
comment: "#$UniversalQuantification is the logical feature of allowing the use of ``for all'' in logical statements.".
termStrings: "universal restrictions" "universal quantification".

Concept: LimitedExistentialQuantification.
isa: LogicFeature.
comment: "#$LimitedExistentialQuantification is the logical feature of allowing the use of ``there exists'' (at least one) in logical statements without restricting membership in a class.".

Concept: RoleRestriction.
isa: LogicFeature.
comment: "#$RoleRestriction is the logical feature of allowing the specification that the instances of a specific class have a restricted range (possibly a single value) for a given binary predicate.".

Concept: FullExistentialQuantification.
isa: LogicFeature.
comment: "#$FullExistentialQuantification is the logical feature of allowing the use of ``there exists (at least one) X such that X is an instance of''.".
f: (partOf LimitedExistentialQuantification FullExistentialQuantification).

Concept: TransitiveRoles.
isa: LogicFeature.
comment: "#$TransitiveRoles is the logical feature of allowing the specification of #$TransitiveBinaryPredicates.".

Concept: FunctionalProperties.
isa: LogicFeature.

comment: "#$FunctionalProperties is the logical feature of allowing the specification of #$FunctionalBinaryPredicates.".
nameStrings: "functional properties" "F".

Concept: ConceptUnion.
isa: LogicFeature.
comment: "#$ConceptUnion is the logical feature of allowing the specification of the union of two classes.".
nameStrings: "concept union" "U".

Concept: ComplexConceptNegation.
isa: LogicFeature.
comment: "#$ComplexConceptNegation is the logical feature of allowing negation of non-atomic terms.".
nameStrings: "C" "complex concept negation".

Concept: BinaryRelationHierarchy.
isa: LogicFeature.
comment: "#$BinaryRelationHierarchy is the logical feature of allowing specification of a hierarchy of binary relations.".
nameStrings: "H" "role hierarchy" "subproperties" "binary relation hierarchy".

Concept: SubRelationHierarchy.
isa: LogicFeature.
comment: "#$SubRelationHierarchy is the logical feature of allowing specification of a hierarchy of subrelations not resticted to binary relations.".
nameStrings: "subproperties" "subrelation hierarchy" "relation hierarchy".

Concept: IrReflexivity.
isa: LogicFeature.
comment: "#$IrReflexivity is the logical feature of allowing specification of reflexive and irreflexive binary relations.".
nameStrings: "reflexivity and irreflexivity" "(ir)reflexivity" .

Concept: BinaryPredicateDisjointness.
isa: LogicFeature.
comment: "#$BinaryPredicateDisjointness is the logical feature of allowing specification that two binary relations are incompatable.".
nameStrings: "role disjointness" "binary predicate disjointness" "binary predicate incompatability" .

Concept: EnumeratedClasses.
isa: LogicFeature.

comment: "#$EnumeratedClasses is the logical feature of allowing specification of classes with enumerated instances.".
nameStrings: "O" "nominals" "enumerated classes".

Concept: InverseProperties.
isa: LogicFeature.
comment: "#$InverseProperties is the logical feature of allowing specification of two binary predicates being inverses of each other, i.e., (PRED1 X Y) <=> (PRED2 Y X).".
nameStrings: "I" "inverse properties".

Concept: CardinalityRestrictions.
isa: LogicFeature.
comment: "#$CardinalityRestrictions is the logical feature of allowing specification of a class, all of whose instances have a specified limited number of fillers of the range of a given binary predicate.".
nameStrings: "N" "cardinality restrictions".

Concept: QualifiedCardinalityRestrictions.
isa: LogicFeature.
comment: "#$QualifiedCardinalityRestrictions is the logical feature of allowing specification of a class, all of whose instances have a specified limited number of fillers of the range of a given binary predicate which are instances of a specified type.".
f: (partOf CardinalityRestrictions QualifiedCardinalityRestrictions).
nameStrings: "Q" "qualified cardinality restrictions".

Concept: DatatypeProperties.
isa: LogicFeature.
comment: "#$DatatypeProperties is the logical feature of allowing specification of predicates whose range is a set of datatypes such as numbers, character strings, or some restricted range thereof.".
nameStrings: "(D)" "datatype properties" "data values" "data types".

Concept: FirstOrderImplications.
isa: LogicFeature.
comment: "#$Implications is the logical feature of allowing specification of rules of the form (forAll (X,...) (implies (and SENTENCE1 ...) CONCLUSION_SENTENCE)) with the variables X,... having their values restricted to instances of #$Individual.".
nameStrings: "first-order implications".

Concept: QuantificationOverClasses.
isa: LogicFeature.

comment: "#$QuantificationOverClasses is the logical feature of allowing quantification (forAll or thereExists with the values of the variables (X,...) being permitted to be classes in addition to being instances of #$Individual.".
nameStrings: "quantification over classes".

Concept: QuantificationOverPredicates.
isa: LogicFeature.
comment: "#$QuantificationOverPredicates is the logical feature of allowing quantification (forAll or thereExists with the values of the variables (X,...) being permitted to be predicates in addition to being instances of #$Individual.".
nameStrings: "QuantificationOverPredicates".

Concept: QuantificationOverFunctions.
isa: LogicFeature.
comment: "#$QuantificationOverPredicates is the logical feature of allowing quantification (forAll or thereExists with the values of the variables (X,...) being permitted to be functions in addition to being instances of #$Individual.".
nameStrings: "QuantificationOverFunctions".

Concept: LogicSystem.
isa: ObjectType.
genls: NormativeSpecification.
comment: "".

Concept: logicalFeatureOfSystem.
isa: BinaryPredicate.
arg1Isa: LogicSystem.
arg2Isa: LogicFeature.
comment: "(#$logicalFeatureOfSystem LOGIC FEATURE) means that the #$LogicSystem LOGIC has the #$LogicFeature FEATURE.".

Constant: DescriptionLogic.
isa: ObjectType.
genls: LogicSystem.
comment: "Description logic is a family of logic systems. Description logics are more expressive than propositional logic but less expressive than first-order predicate logic.".

Constant: ALCDescriptionLogic.
isa: DescriptionLogic.
logicalFeatureOfSystem: AtomicNegation ConceptIntersection
  LimitedExistentialQuantification UniversalQuantification.

Constant: SHIQDescriptionLogic.

isa: DescriptionLogic.
logicalFeatureOfSystem: AtomicNegation ConceptIntersection ComplexConceptNegation
LimitedExistentialQuantification UniversalQuantification TransitiveRoles
BinaryRelationHierarchy QualifiedCardinalityRestrictions.

Constant: ALCOINDescriptionLogic.
isa: DescriptionLogic.
logicalFeatureOfSystem: AtomicNegation ConceptIntersection ComplexConceptNegation
LimitedExistentialQuantification UniversalQuantification EnumeratedClasses
InverseProperties CardinalityRestrictions.

Constant: SHOINDDescriptionLogic.
isa: DescriptionLogic.
logicalFeatureOfSystem: AtomicNegation ConceptIntersection ComplexConceptNegation
LimitedExistentialQuantification UniversalQuantification EnumeratedClasses
InverseProperties CardinalityRestrictions TransitiveRoles DatatypeProperties
BinaryRelationHierarchy.

Constant: SROIQDDescriptionLogic.
isa: DescriptionLogic.
logicalFeatureOfSystem: AtomicNegation ConceptIntersection ComplexConceptNegation
UniversalQuantification LimitedExistentialQuantification TransitiveRoles
EnumeratedClasses InverseProperties DatatypeProperties IrReflexivity
QualifiedCardinalityRestrictions BinaryPredicateDisjointness.

Constant: SHIFDDescriptionLogic.
isa: DescriptionLogic.
logicalFeatureOfSystem: AtomicNegation ConceptIntersection ComplexConceptNegation
UniversalQuantification LimitedExistentialQuantification TransitiveRoles
BinaryRelationHierarchy InverseProperties FunctionalProperties DatatypeProperties.

Constant: FirstOrderLogic.
isa: LogicSystem.
logicalFeatureOfSystem: AtomicNegation ConceptIntersection ComplexConceptNegation
UniversalQuantification FullExistentialQuantification TransitiveRoles
InverseProperties IrReflexivity QualifiedCardinalityRestrictions
BinaryPredicateDisjointness FirstOrderImplications
EnumeratedClasses DatatypeProperties .

Constant: SecondOrderLogic.
isa: LogicSystem.

logicalFeatureOfSystem: AtomicNegation ConceptIntersection ComplexConceptNegation
UniversalQuantification FullExistentialQuantification TransitiveRoles
InverseProperties IrReflexivity QualifiedCardinalityRestrictions
BinaryPredicateDisjointness FirstOrderImplications
QuantificationOverClasses QuantificationOverPredicates QuantificationOverFunctions
EnumeratedClasses .

; ==========

; TemporalObjectType ; #$TemporalObjectType
; TemporalStuffType ; #$TemporalStuffType
; SpatialObjectType ; #$ExistingObjectType
; SpatialStuffType ; #$ExistingStuffType

====

OOPS!

Oops! Catalog (María Poveda-Villalón, Mari Carmen Suárez-Figueroa and Asunción Gómez-Pérez)
also: Poveda-Villalon, M. et al "Validating Ontologies with OOPS!" (2012)
also" http://oa.upm.es/5413/1/A_DOUBLE_CLASSIFICATION_OF_COMMON_PITFALLS_IN_ONTOLOGIES.pdf

- **Human understanding** dimension refers to whether the ontology provides enough information so that it can be understood from a human point of view. This aspect is highly related to the ontology documentation and clarity of the code.
- **Logical consistency** dimension refers to whether (a) there are logical inconsistencies or (b) there are parts of the ontology that could potentially lead to an inconsistency but they cannot be detected by a reasoner unless the ontology is populated.
- **Modelling issues** dimension refers to whether the ontology is defined using the primitives provided by ontology implementation languages in a correct way, or whether there are modelling decision that could be improved.
- **Ontology language specification** dimension refers to whether the ontology is compliant (e.g., syntax correctness) with the specifications of the ontology language used to implement the ontology
- **Real world representation** dimension refers to how accurately the ontology represents the domain intended for modelling. This dimension should be checked by humans (e.g., ontology engineers and domain experts).
- **Semantic applications** dimension refers to whether the ontology is fit for the software that uses it, for example checking availability, format compatibility, etc.

Pitfall Catalog Table

(page 5 of paper, slide 8 of presentation)

P1. Creating polysemous elements: an ontology element whose name has different meanings is included in the ontology to represent more than one conceptual idea. For example, the class “Theatre” is used to represent both the artistic discipline and the place in which a play is performed.
P2. Creating synonyms as classes: several classes whose identifiers are synonyms are created and defined as equivalent. As an example we could define “Car”, “Motorcar” and “Automobile” as equivalent classes. Another example is to define the classes “Waterfall” and “Cascade” as equivalents. This pitfall is related to the guidelines presented in [X7X] which explain that synonyms for the same concept do not represent different classes.
P3. Creating the relationship “is” instead of using “subclassOf”, “instanceOf” or “sameIndividual”: the “is” relationship is created in the ontology instead of using OWL primitives for representing the subclass relationship (“subclassOf”), the membership to a class (“instanceOf”), or the equality between instances (“sameAs”). An example of this type of pitfall is to define the class “Actor” in the following way „Actor ≡ Person interprets.Actuation is.Man“. This pitfall is related to the guidelines for understanding the “is-a” relation provided in [X7X].
P4. Creating unconnected ontology elements: ontology elements (classes, relationships or attributes) are created with no relation to the rest of the ontology. An example of this type of pitfall is to create the relationship “memberOfTeam” and to miss the class representing teams; thus, the relationship created is isolated in the ontology.
P5. Defining wrong inverse relationships: two relationships are defined as inverse relations when actually they are not. For example, something is sold or something is bought; in this case, the relationships “isSoldIn” and “isBoughtIn” are not inverse.
P6. Including cycles in the hierarchy [X4X, X7X]: a cycle between two classes in the hierarchy is included in the ontology, although it is not intended to have such classes as equivalent. That is, some class A has a subclass B and at the same time B is a superclass of A. An example of this type of pitfall is represented bythe class “Professor” as subclass of “Person”, and the class “Person” as subclass of “Professor”.
P7. Merging different concepts in the same class: a class is created whose identifier is referring to two or more different concepts. An example of this type of pitfall is to create the class “StyleAndPeriod”, or “ProductOrService”.
P8. Missing annotations: ontology terms lack annotations properties. This kind of properties improves the ontology understanding and usability from a user point of view.
P9. Missing basic information: needed information is not included in the ontology. Sometimes this pitfall is related with the requirements in the ORSD [X12X, X13X] that are not covered by the ontology. Other times it is related with knowledge that could be added to the ontology in order to make it more complete. An example of this type of pitfall is to create the relationship “startsIn” to represent that the routes have a starting point in a particular location; and to miss the relationship “endsIn” to show that a route has an end point. Another example is to create the relationship “follows” when modelling order relations; and do not create its inverse relationship “precedes”.

P10. Missing disjointness [X4X, X7X, X11X]: the ontology lacks disjoint axioms between classes or between properties that should be defined as disjoint. For example, we can create the classes “Odd” and “Even” (or the classes “Prime” and “Composite”) without being disjoint; such representation is not correct based on the definition of these types of numbers.
P11. Missing domain or range in properties: relationships and/or attributes without domain or range (or none of them) are included in the ontology. There are situations in which the relation is very general and the range should be the most general concept “Thing”. However, in other cases, the relations are more specific and it could be a good practice to specify its domain and/or range. An example of this type of pitfall is to create the relationship “hasWritten” in an ontology about art in which the relationship domain should be “Writer” and the relationship range should be “LiteraryWork”. This pitfall is related to the common error when defining ranges and domains described in [X11X].
P12. Missing equivalent properties: when an ontology is imported into another, classes that are duplicated in both ontologies are normally defined as equivalent classes. However, the ontology developer misses the definition of equivalent properties in those cases of duplicated relationships and attributes. For example, the classes “CITY” and “City” in two different ontologies are defined as equivalent classes; however, relationships “hasMember” and “has-Member” in two different ontologies are not defined as equivalent relations.
P13. Missing inverse relationships: there are two relationships in the ontology that should be defined as inverse relations. For example, the case in which the ontology developer omits the inverse definition between the relations “hasLanguageCode” and “isCodeOf”, or between “hasReferee” and “isRefereeOf”.
P14. Misusing “allValuesFrom” [X11X]: this pitfall can appear in two different ways. In the first, the anomaly is to use the universal restriction (“allValuesFrom”) as the default qualifier instead of using the existential restriction (“someValuesFrom”). This means that the developer thinks that “allValuesFrom” implies “someValuesFrom”. In the second, the mistake is to include “allValuesFrom” to close off the possibility of further additions for a given property. An example of this type of pitfall is to define the class “Book” in the following way „Book ≡ producedBy.Writer uses.Paper‟ and closing the possibility of adding “Ink” as an element used in the writing.
P15. Misusing “not some” and “some not” [X11X]: to mistake the representation of “some not” for “not some”, or the other way round. An example of this type of pitfall is to define a vegetarian pizza as any pizza which both has some topping which is not meat and also has some topping which is not fish. This example is explained in more detail in [X11].
P16. Misusing primitive and defined classes [X11X]: to fail to make the definition “complete‟ rather than „partial‟ (or „necessary and sufficient‟ rather than just „necessary). It is critical to understand that, in general, nothing will be inferred to be subsumed under a primitive class by the classifier. This pitfall implies that the developer does not understand the open world assumption. A more detailed explanation and examples can be found in [X11X].
P17. Specializing too much a hierarchy: the hierarchy in the ontology is specialized in such a way that the final leaves cannot have instances, because they are actually instances and should have been created in this way instead of being created as classes. Authors in [X7X] provide guidelines for distinguishing between a class and an instance when modelling hierarchies.
An example of this type of pitfall is to create the class “RatingOfRestaurants” and the classes “1fork”, “2forks”, and so on, as subclasses instead of as instances. Another example is to create

the classes “Madrid”, “Barcelona”, “Sevilla”, and so on as subclasses of “Place”. This pitfall could be also named “Individuals are not Classes”.

P18. Specifying too much the domain or the range [X7X, X11X]: not to find a domain or a range that is general enough. An example of this type of pitfall is to restrict the domain of the relationship “isOfficialLanguage” to the class “City”, instead of allowing also the class “Country” to have official language or a more general concept such as “GeopoliticalObject”.

P19. Swapping intersection and union: the ranges and/or domains of the properties (relationships and attributes) are defined by intersecting several classes in cases in which the ranges and/or domains should be the union of such classes. An example of this type of pitfall is to create the relationship “takesPlaceIn” with domain “OlympicGames” and with range the intersection of the classes “City” and “Nation”. Another example can be to create the attribute “Name” for the classes “City” and “Drink” and to define its domain as the intersection of both classes. This pitfall is related to the common error that appears when defining ranges and domains described in [X11X] and also related to the guidelines for defining these elements provided in [X7X].

P20. Swapping Label and Comment: the contents of the Label and Comment annotation properties are swapped. An example of this type of pitfall is to include in the Label annotation of the class “Crossroads” the following sentence "the place of intersection of two or more roads"; and to include in the Comment annotation the word „Crossroads".

P21. Using a miscellaneous class: to create in a hierarchy a class that contains the instances that do not belong to the sibling classes instead of classifying such instances as instances of the class in the upper level of the hierarchy. This class is normally named “Other” or “Miscellaneous”. An example of this type of pitfall is to create the class “HydrographicalResource”, and the subclasses “Stream”, “Waterfall”, etc., and also the subclass “OtherRiverElement”.

P22. Using different naming criteria in the ontology: no naming convention is used in the identifiers of the ontology elements. Some notions about naming conventions are provided in [X7X]. For example, we can name a class by starting with upper case, e.g. “Ingredient”, and its subclasses by starting with lower case, e.g. “animalorigin”, “drink”, etc.

P23. Using incorrectly ontology elements: an ontology element (class, relationship or attribute) is used to model a part of the ontology that should be modelled with a different element. A particular case of this pitfall regarding to the misuse of classes and property values is addressed in [X7X]. An example of this type of pitfall is to create the relationship “isEcological” between an instance of “Car” and the instance “Yes” or “No”, instead of creating the attribute “isEcological” whose range is Boolean.

P24. Using recursive definition: an ontology element is used in its own definition. For example, it is used to create the relationship “hasFork” and to establish as its range the following "the set of restaurants that have at least one value for the relationship “hasFork”

Human understanding

P1. Creating polysemous elements

P2. Creating synonyms as classes

P7. Merging different concepts in the same class
P8. Missing annotations
P11. Missing domain or range in properties
P12. Missing equivalent properties
P13. Missing inverse relationships
P19. Swapping intersection and union
P20. Misusing ontology annotations
P22. Using different naming criteria in the ontology

Modelling issues

P2. Creating synonyms as classes
P3. Creating the relationship “is” instead of using "rdfs:subClassOf", "rdf:type" or "owl:sameAs"
P4. Creating unconnected ontology elements
P5. Defining wrong inverse relationships
P6. Including cycles in the hierarchy
P7. Merging different concepts in the same class
P10. Missing disjointness
P17. Specializing too much a hierarchy
P11. Missing domain or range in properties
P12. Missing equivalent properties
P13. Missing inverse relationships
P14. Misusing "owl:allValuesFrom"
P15. Misusing “not some” and “some not”
P18. Specifying too much the domain or the range
P19. Swapping intersection and union
P21. Using a miscellaneous class
P23. Using incorrectly ontology elements
P24. Using recursive definition
P25. Defining a relationship inverse to itself
P26. Defining inverse relationships for a sym-metric one
P27. Defining wrong equivalent relationships
P28. Defining wrong symmetric relationships
P29. Defining wrong transitive relationships

Logical consistency

P5. Defining wrong inverse relationships
P6. Including cycles in the hierarchy
P14. Misusing "owl:allValuesFrom"
P15. Misusing “not some” and “some not”
P18. Specifying too much the domain or the range
P19. Swapping intersection and union
P27. Defining wrong equivalent relationships
P28. Defining wrong symmetric relationships
P29. Defining wrong transitive relationships

Real world representation

P9. Missing basic information

P10. Missing disjointness

====
====
====
====
====

Ontology Evaluation with Respect to Specific Requirements
Track A Synthesis, Obrst (this is captured above)
Also covered in West, above
In Duque-Ramos, et al, above, re Functional adequacy
Track B Synthesis, Longstreth & Schneider
Extrinsic aspects to be considered include,

- requirements and their verification
- how metrics can be derived from requirements
- how 'good' requirements relevant to ontology can be crafted
- fitness for purpose
- …

Bennett

Matthews

Vizedom

Luciano

capability questions

From several slide sets in session 7

Ontology requirements that may be derived from business requirements

- Lexical
    - Multiple Lexicons mapped to same ontology
    - Lexical coverage for particular language, locality, or user community (technical background, organization-based, shared activity-based, at particular time)
    - Lexical detail on expressions (e.g. POS, case, …) supporting natural language processing (stem, morph, etc)
- Logical (expressibility of supporting structures for; existence of & compatibility wit reasoners for;...)
    - Rules
    - Quantification
    - Variables

    - Variable that range over [classes|relations|statements]
    - Deductive reasoning
    - monotonic - non-monotonic reasoning
    - Probabilistic
    - modal
- Specific content
    - Sufficient to express and answer particular questions or kinds of questions
    - Sufficient to support semantic [matching|disambiguation|indexing|retrieval] over a particular corpus or stream
    - Sufficient to express and support capture of typical data in a stream
    - Sufficient to support modeling of simple or complex objects in a particular domain
    - Sufficient to support [manual|semi-automated|automated] classification of objects in a particular domain
    - Sufficient to support classification of information objects in a particular domain
- Specific alignment
    - with standards
    - with other ontologies
    - with regulations
- provenance and tracing
    - included provenance of assertions
    - included provenance of inferences
    - support expression of provenance of new assertions
    - support expression of provenance of new inferences
-